

respectively, of the Package Implementation. The Package Instance also includes a pointer to the Package Implementation of which the Package Instance is an instance.

[0519] The Packages Implementation, Models Implementation, Constraint Implementations, and Attribution Implementations create a Packages Handler, a Models Handler, a Constraints Handler, and an Other Attributes Handler of the Package Impl Generator.

[0520] The Handler, Models Handler, Constraints Handler, and Other Attributes Handler of the Package Impl Generator are read by the Packages Descriptor, as indicated by connector WW, the Models Descriptor, as indicated by the connector WW, the Constraint Descriptors, as indicated by the connector XX, and by the Other Attributes, as indicated by the connector YY, respectively, for the Package Descriptor.

[0521] FIG. 33 also shows Attribute Instances including Packages, Models and Other Attributes.

[0522] In FIGS. 5-34, there are automatic conversions from UML, such as text descriptions, to the descriptor objects in the Meta-Implementation Layer of the Component Integration Engine. Although not shown in the embodiment of the component integration engine of the present invention of FIGS. 5-34, there may be zero or more handlers to convert various source data to the appropriate descriptor. For each handler there exists a Serializer to convert from a descriptor that that source data format. The user may also directly create a descriptor without using a handler.

[0523] A component integration engine of the present invention may be used to create connections between objects dynamically at run-time instead of creating object connections at compile-time. Since object connections are not created at compile time, each object must operate correctly independently of the connections that might be established later. This statement is equivalent to saying that code is more reliable. Creating connections at run-time rather than compile-time offers significantly more flexible functionality at run-time. Using metadata to perform these connections makes connections self-describing. Flexible functionality that is easy to understand (due to its self-describing nature) leads to lower maintenance costs.

[0524] Using metadata in at least some embodiments of the component integration engine of the present invention to perform connections between components prevents unexpected access from one component to another component. This restriction on access reduces unexpected side effects between components reducing the types of errors that can occur and limiting the scope of an error if it should occur. In fact, errors are unlikely to cross the metadata connection. These restrictions significantly reduce testing and debugging.

[0525] In some embodiments, the component integration engine of the present invention may check security restrictions before performing connections between objects. This allows security to be centralized in the component integration engine, meaning components need to perform little or no additional security checks, while still providing security throughout the entire application.

[0526] In some embodiments, the component integration engine of the present invention may identify performance

bottlenecks at run-time by logging performance characteristics for each component or component connection. Components can be dynamically reconfigured or replaced at run-time to eliminate bottlenecks. Since the creation of object connections can be performed at runtime and can be performed through compiled metadata that is negligibly slower than compiled code, smaller components can be developed and connected at run-time. Smaller components represent smaller units of functionality and smaller units of data, which leads to a reduction in the proliferation of models and increases the reusability of code (non-redundant code). This propensity of the system to produce smaller units of code, which do not require extensive inheritances, leads to a reduction in development (which correspondingly leads to a reduction in testing and debugging).

[0527] The component integration engine of the present invention describes a software application in order to dynamically create that software application without programming is a new category of software. The only similar software in existence is a system of metadata used to describe relational data structures for storage and retrieval (a database). Overcoming implementation difficulties and discovering a high-performing, secure, scalable mechanism to assemble components of any type to affect data of any type in an effort to perform a software task of any type requires several novel software patterns and requires the use of compiled metadata.

[0528] In some embodiments, the component integration engine of the present invention may include Hierarchical Model View Controller using events based on metadata (an HMVC 4.6.2 pattern). The model-view-controller pattern has been redesigned to use a “model”—“model controller”—“view controller”—“view” pattern each part of which is allowed to be a hierarchy of objects. Communications occur only between adjacent parts. Communications between the model controller and view controller only occur at the top-most level of the hierarchy, instead of the traditional MVC or HMVC patterns which allow communication between any parts in the system and at any level. Adding this restriction between the model and view controllers allows greater distribution to occur by inserting a “forward” between them (leading to the pattern of “model”—“model controller”—“server-side forward”—“client-side forward”—“view controller”—“view”). Events define the values they carry using a definition. Upon initial registration of an event listener, the metadata definitions are passed to the listener followed by the current values. This innovation allows listeners to more correctly respond to events by adjusting to differences in the metadata, or using the metadata to more fully constrain values.

[0529] In another embodiment, the present invention provides a two-part graphical user interface design. The first part is the use of metadata as part of the events sent between the model and view. The second part is the change in the structure used to handle these events. The events sent between the model and view relate to changes in values that need to be displayed on the view or adjusted on the model. Different events signify a request for an action to occur, a failure occurring, a change in the display, a change in selection, a change in cursor position, etc. In order to make event handling and event registration easier, the wide variety of different events have been consolidated into a single type of event: an application event. In order to still support the